ISSN 0265-2919

R 80p

THE HOME COMPUTER COURSE 24

MASTERING YOUR HOME COMPUTER IN 24 WEEKS

461 Computers and Gambling
464 MACHINE CODE
466 Computer of the Future
468 The Fifth Generation
470 Research Machines 380Z
472 Sound and Light
474 Basic Programming
478 History of the Micro

An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Hardware Focus

Research Machines 380Z High resolution graphics make this the most popular microcomputer in schools and colleges 470

Software

Calculated Risk Can a home computer help you win the football pools? We investigate the links between computers and gambling 461

Basic Programming

Language Lab We round off our look at the Basic programming language by considering its overall strengths and weaknesses 474

Insights

Futuristic What major developments are likely for home computers in the immediate future? 466

Passwords To Computing

Assembly Line Assembly language translates directly to machine code, but is much easier to read and develop 464

Generation Gap We look at the implications of the Japanese research project to create the Fifth Generation of computers 468

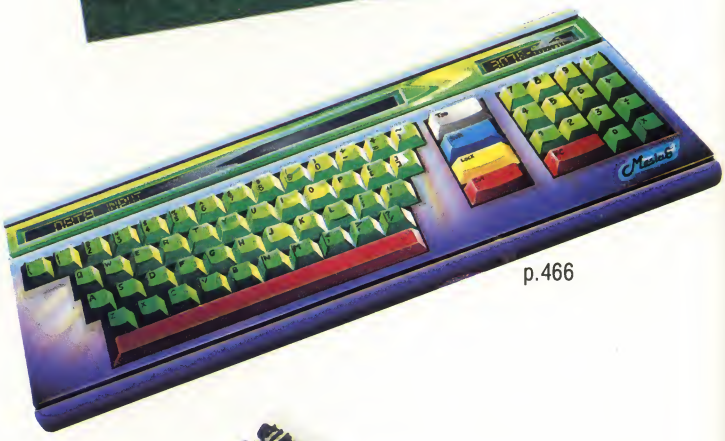
Pioneers In Computing

Firm Foundations Six key individuals laid the foundations for the way in which the microcomputer industry has developed 478

Sound And Light

Sounds Ideal . . . Guiding Light We see why the Commodore 64's Basic doesn't do justice to its superb sound features, and in a second look at the Atari machines, we investigate Player-Missile graphics 472

p.461


p.466

p.478

COVER PHOTOGRAPHY BY IAN McKINNELL

Editor Richard Pawson; **Consultant Editor** Gareth Jefferson; **Art Director** David Whelan; **Production Editor** Catherine Cardwell; **Staff Writer** Roger Ford; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Art Assistant** Liz Dixon; **Sub Editors** Robert Pickering, Keith Parish; **Researcher** Melanie Davis; **Contributors** Tim Heath, Brian Morris, Lisa Kelly, Steven Colwill, Martin Andrewartha, Peter Jackson, Julian Allason, Richard King; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1; **© 1984 by Orbis Publishing Ltd: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95
How to obtain your copies of HOME COMPUTER COURSE – Copies are obtainable by placing a regular order at your newsagent.
Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.
How to obtain binders for HOME COMPUTER COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 4, 5 and 6. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595. Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.
Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

Calculated Risk

REX FEATURES LTD

A Day At The Races
Horse racing is an interesting application of home computers, but strictly for those who know what they are doing. At least one breeder makes use of an Apple II microcomputer to keep a database of the lineage of all his horses, thereby using the computer to attempt to produce winners

Computers have many applications in the world of gambling. Pools programs are even available for home computers

Gambling is all about probability, although most gamblers would prefer to say that it was about winning. Such an assertion is unfounded, of course, for the vast majority of gamblers lose consistently, and sometimes heavily. They do so largely because the odds are against them and in favour of the casino, the bookmaker and the pools operator. To evaluate whether computers could help to redress the balance we must first consider these odds.

Stripped of their outward trappings, all games of chance boil down to betting on the outcome of a chance event. Usually this is generated by some random device, such as a ball spun in a roulette wheel, or a card drawn from a carefully shuffled pack. If the parameters — the number of cards, say — are known, probability theory permits certain predictions to be made about the likelihood of the chance event occurring. For example, the roulette wheel used in British casinos has 37 slots numbered from 0 to 36. There are thus 18 odd and 18 even numbered slots into which the ball might fall, plus the zero. The probability of the ball landing in a slot bearing an odd number can therefore be expressed as 18/37, or 0.4864864, or a little better than 48.6%. This is somewhat less than the evens chance of a coin landing head upwards; the difference, accounted for by the presence of the zero slot, representing the house's 'edge' or profit margin.

It is this 'edge' that makes games of chance, and most other forms of gambling, so unrewarding. Despite the claims occasionally advanced by purveyors of gambling systems, there is nothing that a computer can do to improve the basic odds in a given game. Indeed, one leading authority, Professor Hans Sagan, has calculated that in the long run it is impossible to win at any casino game, except possibly blackjack (also known as *vingt-et-un*).

Such theoretical objections have failed to discourage enthusiastic inventors, and home computer owners are now being offered a variety of allegedly fool-proof gambling systems, some of which even appear to work. In the case of casino gaming systems, these almost invariably prove to be variations on 'doubling up', a procedure that suffers from the disadvantage of requiring an infinite amount of stake money to succeed. There is a further problem as well: though not technically illegal, the management of casinos both in the UK and USA will not allow computers of any kind to be used. Computers are arguably of greater assistance where skill or strategy are involved. The computer can be used to instil the necessary discipline into the player, and to act as a memory aid. However, the value of such assistance tends to be in inverse proportion to the skill of the player.

Unfortunately, the skill element involved in most popular forms of gambling is minimal.

Football pools offer a case in point. The most sophisticated pools prediction program is Professor Frank George's celebrated F4 Football Forecast, which is available in versions suitable for most popular home computers. Based on ten years of statistical analysis, the program attributes a value to the average performance of each team. When adjusted by weighting for long term form (drawn from the league tables), short term form, and last match result, a comparison of these performance figures enables the program to predict the probable result of a given match.

With Liverpool playing at home to Brighton, the result might be a foregone conclusion, but the program comes into its own when predicting the outcome of a game between more evenly matched teams. This is not to say that this system is a sure-fire winner. Statistical analysis suggests that using Professor George's program approximately trebles the probability of success. 'I concede that the chances against winning are still huge, but surely it is better to gamble as intelligently as you can?' he asks.

Even given this assistance, the odds remain unfavourable. Littlewoods, one of the largest of the pools promoters, say that none of their big prizes has ever been won by anyone who had used a personal computer. 'If there was a system that really worked, we would know about it, and there isn't,' states Littlewoods' spokesman, Tony Hodges. Although entry marking is done using special automatic machines, the company itself use computers only for record keeping.

Horse-racing appears to offer, if anything, even greater scope to the programmer. One Darlington schoolboy has created a home computer program to forecast winners. Originally written for the Sinclair ZX-81, and now upgraded to run on the Spectrum, David Stewart's program has been successful a number of times. Although David's tips are broadcast by several of the BBC local radio stations, he has not amassed a personal fortune.

DAVID W. HAMILTON

Wheeling And Dealing
On a roulette wheel it is the number zero that provides the profit for the casino. Nothing can be done to improve the player's basic odds, so programs devised for such games must concentrate on systems for betting

Perhaps significantly, racing professionals have largely rejected the use of computers. Official handicapping is still done manually by the Jockey Club (although the data is stored on computer). Timeform, the bible of the racegoer, also compiles most of its data manually. 'We only use the computer for calculating the standard time figures for each horse, taking into account wind

Pooled Resources
Several packages are available for home computers that claim to improve your chances at winning the pools, and a great many programmers have attempted to write their own. The better ones make use of a vast database of information on previous matches, and can prove valuable in predicting the outcome of marginal matches. As with all forms of Computer Assisted Gambling (CAG) such programs can only increase your chances marginally, and packages thus come complete with disclaimers from the suppliers

deflections,' explains the publication's Managing Director, Reginald Griffin. 'There is no such thing as a true computerised handicapping system available anywhere. The problem is that computers simply can't cope with the extraordinary results that crop up every day.'

Computers are increasingly used on the other side of the betting shop counter, although not for calculating the odds. Staff employed by the large bookmaking chains are trained to use special dedicated calculators for computing the returns on bets. The credit side of the business is becoming increasingly computerised. A punter with an account at one of the chain bookmakers can simply telephone his bet direct to their computer centre. The details are keyed in and the account debited with the value of the bet. If the chosen horse performs as anticipated, the winnings are calculated and credited to the customer's account file.

Most bookies are sceptical of computer systems. 'No one has ever come up with one that wins consistently — or we wouldn't be here' says William Hill's spokesman, Graham Sharpe. Nonetheless, it was his firm that staged one of the most extraordinary and controversial computer simulations of all time. Form details of the classic Derby winners of the past were incorporated into a specially commissioned program. Newspaper readers were then invited to predict the first six. The controversy arose over the placing of the great Italian horse, Ribop, which never lost a race. The computer placed it fourth!

Perhaps the most famous 'gambling' computer of all is ERNIE (Electronic Random Number Indicator Equipment), the machine that picks the winning premium bond prize numbers. It is arguable whether ERNIE is really a computer at

all; but although it is not programmable, it does execute a program. The machine was developed by Plessey in 1973 to replace the original machine, built in 1957. Its function is to randomly generate some 200,000 numbers and write them onto magnetic tape. These numbers are generated from a series starting at the lowest premium bond serial number ever issued, and ending with the highest. The tape is then loaded into an ICL mainframe computer, which compares the numbers with a tape listing the numbers of those bonds that have already been repaid. Once these ineligible numbers have been eliminated, the computer can print out prize warrants and letters to the winners.

Since commissioning, the two ERNIEs have generated the numbers for 22.2 million prizes worth £1,181,843,400. Sounds good? Not really; the chance of a bond winning a prize in the monthly draw is just one in 15,000.

Another often encountered game of chance appears daily in our newspapers. The chances of winning one of the million pound prizes offered in newspaper promotions is even more remote. The numbers on the cards distributed to readers are 12 digits long. A 12 digit sequence running from 000000000000 to 999999999999 offers a million million separate combinations. Statistically, the odds against a particular number coming up on a given morning would be slightly better than a million million to one, since the newspaper concerned publishes two numbers each day. On this basis, it must be extremely doubtful whether the newspaper will *ever* have to pay out the big prize.

The situation can best be visualised by imagining a bag containing two and a half million white balls, representing the competitors (the number of cards in circulation), and a million million black balls for the total number of possible combinations. Needless to say, the chances of pulling a white one out the first time are pretty remote. Moreover, the odds do not significantly improve even when a year's worth have been pulled out. Statisticians compute the chances of the newspaper ever having to part with a million pounds at just one in 667.

Twist Of Fate
Pontoon (blackjack or 'vingt-et-un') is available as a game on most home computers, and provides some of the best scope for writing winning programs. The ability to memorise the cards already played increases a player's chances of success, though as casinos won't allow computers at the tables, the celebrated feats have all involved concealed computers (in one case strapped to a player's leg underneath his trousers) or radio links to external machines

Loading The Dice

Gambling's most essential ingredient, random number generation, can be easily simulated on a personal computer. Most versions of BASIC provide a random number generator function. In many cases the numbers so generated are not truly random, however, as the following short program demonstrates:

```
10 LET A = RND
20 LET B = RND
30 LET C = RND
40 PRINT A, B, C
```

In each of the first three lines a supposedly random number is assigned to the variables A, B and C. These are then printed out. This might give the following results (expect yours to differ though):

```
.014007  .964370  .457397
```

But if you re-run the program, most microcomputers will display the same sequence again. What is happening is this: when we ask for RND, the computer responds with the next in a fixed sequence of numbers. Typically this might comprise the one million six-digit fractions between 0.000000 and 0.999999, each occurring once in the complete cycle — but not, of course, in sequence.

Certain BASICS use a slightly different syntax, requiring an expression in parenthesis, called an 'argument'. This takes the form LET A = RND (X). The effect is very similar: RND and RND (X) can both be used in the same way as other variables.

Some BASICS also feature a RANDOMIZE function, which causes the sequence to start at an unpredictable point. Inserting the RANDOMIZE command early on in any program where RND is to be used ensures that a different sequence of numbers will be generated each time the program is RUN.

To simulate the casting of a dice we require integers in the range 1 to 6. It is, however, necessary to eliminate fractions. This is done by using the INTeger function. PRINT INT(6.99) produces the result 6 just as surely as PRINT INT (6.01) does. Anything after the decimal point is entirely discarded.

Since the largest number that RND can generate is .999999 (which, when expressed as an integer, acquires a value of 0), a little multiplication is required. The time honoured formula is:

```
LET A = INT(6*RND)+1
```

We multiply by six because a die has six faces. The 'plus one' is simply to ensure that the results range from 1 to 6, and not from 0 to 5.

Assembly Line

Continuing our introduction to machine code, we look at the many different forms in which programs can be expressed — from binary to Assembly language

One of the conceptual difficulties that most newcomers experience with machine code is that the programs can take various forms. Any data stored in computer memory ultimately takes the form of eight-bit binary numbers. However, when these are listed out on paper, they occupy a lot of space, are difficult to read and remember, and are prone to typing mistakes. So instead we usually make use of hexadecimal numbers. This has the advantage that the contents of any byte can be expressed as a two-digit number, and any address in the computer's memory range (0 to 65535 in decimal) can be represented by four digits.

When we write a hex number on paper we usually precede it with a $ sign to distinguish it from a decimal number, although the sign does not feature in the computer's memory when the program has been entered. Secondly, when an opcode has a two-byte operand (e.g. LDA $3F80) the two bytes are entered into the machine in the opposite order — i.e. the low byte followed by the high byte. In the example given, therefore, the three bytes would be AD (the hex representation of the LDA opcode in 6502 language) followed by 80, followed by 3F. This makes things easier for the processor, but it can be confusing for the user.

Usually a machine code program is printed as a 'hex dump' — a long list of two-digit hexadecimal values. In addition, a starting address will be given (either in hex or decimal) and the first hex value must be loaded into this location, the second into the next location, and so on. Loading can be achieved by means of the BASIC POKE command. If the starting address is $1000 (4096 in decimal) and the hex dump is:

```
AD      (173 in decimal)
80      (128 in decimal)
3F      (63 in decimal)
```

the program can be loaded with the three BASIC statements:

```
POKE 4096,173
POKE 4097,128
POKE 4098,63
```

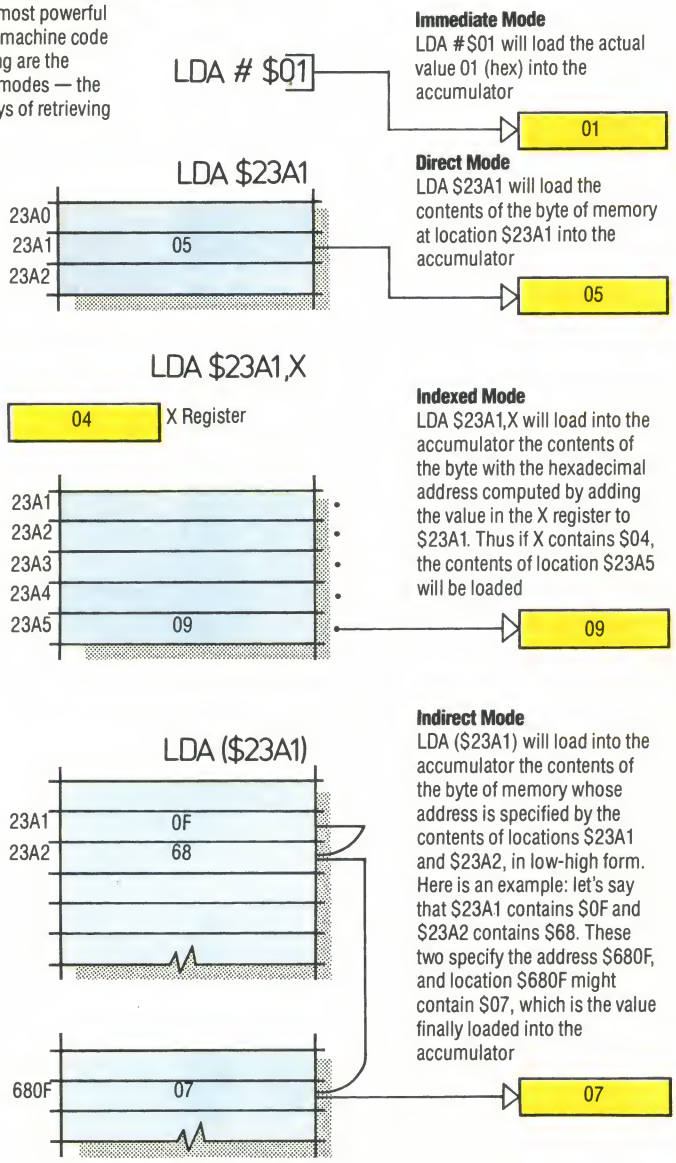
Note how we have to convert all the values from hex to decimal before they can be used in the POKE statement — inside the machine they will be stored in binary.

For longer hex dumps it is normal to use a short BASIC program called a 'machine code loader'. This asks for the start address and then the hex values. As each is entered, the short BASIC routine converts the hex value to decimal, and POKEs it into the next location. Alternatively, the hex dump can be READ by the program from DATA statements.

Once the machine code has been loaded, the BASIC loader program can be dispensed with. It's therefore important to load the machine code somewhere in memory where it won't be 'trampled over' by the BASIC program, nor be

Addressing Modes

Among the most powerful concepts in machine code programming are the addressing modes — the different ways of retrieving data


LDA # $01
Immediate Mode
LDA #$01 will load the actual value 01 (hex) into the accumulator
01
LDA $23A1
23A0
23A1
23A2
05
Direct Mode
LDA $23A1 will load the contents of the byte of memory at location $23A1 into the accumulator
05
LDA $23A1,X
04
X Register
23A1
23A2
23A3
23A4
23A5
09
Indexed Mode
LDA $23A1,X will load into the accumulator the contents of the byte with the hexadecimal address computed by adding the value in the X register to $23A1. Thus if X contains $04, the contents of location $23A5 will be loaded
09
LDA ($23A1)
23A1
23A2
0F
68
680F
07
Indirect Mode
LDA ($23A1) will load into the accumulator the contents of the byte of memory whose address is specified by the contents of locations $23A1 and $23A2, in low-high form. Here is an example: let's say that $23A1 contains $0F and $23A2 contains $68. These two specify the address $680F, and location $680F might contain $07, which is the value finally loaded into the accumulator
07

Longhand/ Shorthand

A machine code program can take on several different forms. It is usually written by the programmer in the form of Assembly language, which uses mnemonics for opcodes and labels for operands, thus:

```
LDA WEIGHT
ADC FUEL
STA WEIGHT
```

We must however specify the addresses of those labels. For example:

```
FUEL = $03EE
WEIGHT = $031F
```

An assembler package would transform this into a hex dump, using a disk drive. 'Pseudo assembly language', as shown below, is less easy to read, but can often be entered into a package called a 'spot assembler', which doesn't need disks.

```
LDA $031F
ADC $03EE
STA $031F
```

A hex dump consists of a starting address (at the left) and the sequence of two-digit hex values as they will appear in memory. Note that an operand like $031F is stored in reverse order (1F 03) and that opcodes have been replaced by the appropriate hex value:

```
19C4 AD 1F 03 6D EE 03 8D 1F 03
```

obliterated by BASIC statements such as NEW.

Most home computers have some BASIC command to tell the machine to stop executing BASIC and begin executing the machine code program that starts at a specific location. One form of this command is SYS 4096 (RETURN), meaning 'transfer control to the system starting at decimal location 4096'; another is CALL $E651, meaning 'call the machine code routine starting at hex location E651'.

The machine code subroutine or program will then execute this system or routine (it may or may not produce any visible results, depending on the nature of the program). If it is correctly written and incorporates the proper terminating procedure, control will be passed back to BASIC. This means, incidentally, that it is possible to call machine code subroutines from several places in the operation of a BASIC program, whenever a function needs to be performed at high speed.

One of the difficulties of programming in machine code is that if you have made a mistake in your code, the computer won't come back with a nice helpful SYNTAX ERROR. It will more than likely 'crash' instead: the machine won't respond to anything you type. This isn't harmful to the computer, but you will have to reset it (or switch the machine off and then on again), and that usually means having to enter the program again from scratch. That's why you can't experiment in machine code as you can in BASIC — the operation of the program must be thoroughly checked on paper before it is entered into the computer.

However, a software device that can assist greatly in the entering and checking of machine code is the 'machine code monitor' (which has nothing to do with a monitor screen). This is built into the ROMs of a few computers but is generally purchased as a cassette or cartridge-based package. A machine code monitor is a simple operating system that will display on the screen the contents of any requested section of memory. These (hex) values can simply be altered or written over, so a monitor is by far the fastest way of entering a hex dump. Moreover, it usually allows you to load and save machine code programs directly onto cassette, without the need for the BASIC loader program. The most advanced machine code utility programs (the machine code equivalent to BASIC tool kits — see page 444) show the contents of each of the processor's internal registers.

Hex dumps are a convenient way of expressing machine code, but they aren't easy to read. Unless you happen to remember the hexadecimal equivalent of all the various opcodes, it's almost impossible to distinguish the opcodes from the operands. So programs are usually written using the three-letter mnemonics that we introduced in the previous article (page 449), and these are then translated into hex using a table of codes from the microprocessor's handbook.

However, a more sophisticated form of machine code monitor will allow you to type in the program in mnemonics, performing the conversions automatically. This is called a 'spot assembler' because it will assemble the mnemonics into numbers on the spot.

This leads us on to the final form in which machine code can be expressed — Assembly language — which not only makes use of mnemonics for the opcodes but can handle names (or labels) instead of hex numbers for the operands. Thus, if location $07B2 contains the current number of missiles fired in a game, we can load this into the accumulator with the instruction:

```
LDA MISSIL
```

At the start of the program we will have to specify the location of MISSIL=$07B2, and that this location should initially contain the value of $09 (nine missiles).

When we have finished developing this program in Assembly language (called the 'source code' of the program), we run a utility program called an assembler. This works through the code, replacing mnemonics and any labels with their hex equivalent, thereby creating a new version called the 'object code'. This code can then be entered into the computer's memory and run. The process is not dissimilar to compiling (see page 84), though in this case there is a one-to-one correspondence between the source and object code.

Assembly language, being a higher-level language than machine code, is considerably easier to write, but there is no loss in performance. However, assembler packages will usually only work with a disk drive, and so are not available to all home computer users.

Opcodes

Here are some more opcodes that a typical microprocessor would feature

JSR

Jump SubRoutine

This function is equivalent to BASIC's GOSUB. JSR $354D will change the contents of the program counter (PC) register so that it executes the code from $354D onwards

RTS

ReTurn from Subroutine

On encountering RTS, the processor will jump back to the location from which the subroutine was called (i.e. equivalent to RETURN in BASIC). RTS has no operand because the return address will have automatically been stored in a special area of memory called the Stack

BMI

Branch if MInus

This is one of several forms of conditional branching in machine code (in BASIC, IF . . . THEN GOTO is a conditional branch). If the result of the last operation resulted in a negative value in the accumulator, program execution will jump to a specified address. BPL specifies Branch if PLus

LDX

LoaD X register

X is another single byte register within the processor, and while it cannot perform arithmetic in the same way as the accumulator, it is used for 'indexed addressing' (see panel). LDX loads a value into X, and STX (STore X) will store it back in memory

INX

INcrement X

By adding 1 to the value of X (DEX — DEcrement X — will subtract 1), and using indexed addressing, it is possible to step through a number of locations in memory, performing the same process on each

Futuristic

Home computers have developed enormously over the last five years, but what will the next five years bring? Compare our ideas with your predictions

What will the home computer of the 1990's look like and how will it function? These are the questions that this section will attempt to answer, by considering in turn some of the main components and systems of tomorrow's machine. Many of the ideas are based on technologies that are just coming on the market (perhaps in other fields than computing), while others represent what we believe to be likely developments.

One of the most fundamental features of our hypothetical design is modularity. Having purchased the base unit, the user will have a wide range of options for expanding the machine. Indeed, the user will virtually be able to design his own machine by selecting this graphics module and that sound facility. Of one thing we can be sure: the rate of change in the computer marketplace will continue to accelerate for many years to come.

1 Keyboard Display

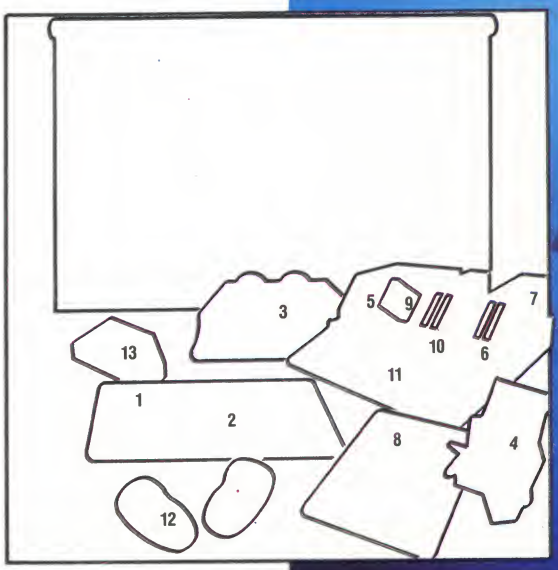
The 32-bit microprocessor's power will allow the display of information in a number of forms simultaneously. For instance, the main screen might show the view from the command seat of a spacecraft, while a subsidiary screen mounted on top of the keyboard/command console might display control information from the cockpit

3 Monitor

Projector televisions have been available since the beginning of the 1980's, but their scope is limited by the light-emitting power of the cathode ray tube. Advances in CRT technology are likely to bring us room-wide projection systems. Early projector televisions had to make use of special curved screens, but the latest models can already focus onto a flat surface

2 Keyboard

Despite the innate inefficiency of the QWERTY keyboard, it is unlikely that serious attempts will be made to establish an alternative layout. Fully sprung typewriter-style keys are by far the most popular — though Hall effect keys, which use magnets instead of springs, are likely to become commonplace. The electronic switches themselves may well be replaced by a system that relies on the keys interrupting a matrix of laser beams



TONY LODGE

4 Alternative Processors

In addition to the main 32-bit processor, it is likely that the micro of the 1990's will be host to additional processors in the form of plug-in modules. Some of the processing — for example, the operation of a particular peripheral or sorting a file of data — can then be 'subcontracted' by the main processor to the most suitable sub-processor. Alternatively, inexpensive plug-in modules could emulate the classic computers of the 1980's, so that software from any other computer could be run without modification

5 Random Access Memory

The 32-bit processor can address up to almost 4,300 million memory locations — a far cry from the 65,536-byte limit imposed by the eight-bit processors that brought microcomputers into the home

6 Communications

While dish aerials for the reception of signals from satellites will be commonplace by the 1990's and most telephone channels will be digitised, rather than relying on analogue signals, there will still be a need to regulate the speed of transmission and reception. These communications controllers will perform some of the control functions of today's modulator/demodulators

7 Power Supply

The increased load and the multiplicity of devices connected to the microcomputer are likely to require a significantly greater power supply than those in use today. It will incorporate smoothing circuits and rechargeable battery back-up, so that mains fluctuations or power failures do not cause data to be lost or corrupted

8 Portable Screen

Flat-screen technology — probably involving a fast-acting liquid crystal matrix and perhaps connected to the central processor by an infra-red (or even microwave) link — may be employed to display text and graphic matter. If this device were touch-sensitive, too, it could double as a menu-selection board and bit-pad or digitiser

9 CDROM

The Compact Disk ROM, which uses a laser beam to read optically-encoded information, is likely to replace conventional ROM cartridges because of its much greater capacity — a typical CDROM will hold four megabytes

10 Floppy Diskettes

By the end of the decade floppy diskettes should have evolved to compete directly with Winchester disks, both in speed and data-packing densities. At the same time they should reduce in diameter to less than the current minimum of 3 ins

11 Front Panel

On the early computers, before the advent of high-level languages and keyboards, programs had to be entered in binary notation by means of the front panel — a line of lights and switches giving the user control over every bit of the address, data and control buses. For experienced machine code enthusiasts, a front panel can still be a useful tool, so this idea might re-emerge on future home computers.

12 Infra-Red Mice

The IBM PC-Junior already makes use of infra-red radiation to transfer data from keyboard to computer without a cable link. This technology could provide the interconnection between all peripherals, including mice, thereby eliminating the 'spaghetti effect'. Both left- and right-handed models will, of course, be available

13 32-Bit Microprocessors

The first 32-bit microprocessor-based home computers appeared in 1983, but were forced to rely on 16- or even eight-bit data buses to maintain compatibility with existing memory and peripheral chips, and thus could not deliver the power they promised. With the introduction of devices such as Motorola's 68032 chip, which offers 32-bit processing and 32-bit data transfer, the speed and data-handling capabilities of these large-capacity microprocessors will become the accepted standard. Many minicomputers costing tens of thousands of pounds have 32-bit processors

Generation Gap

With the introduction of VLSI technology, we are now about to enter the Fourth Generation of computers. But the Japanese are already specifying the Fifth Generation

Old men do not create revolutions, as the saying goes, and the director of the Japanese project to create the Fifth Generation of computers seems to have taken that to heart. In choosing 40 scientists from ten major corporations and government laboratories to work with him at the Institute for New Generation Computer Technology in Tokyo, Dr Kazuhiro Fuchi selected only those under the age of 35. The Institute was founded on 14 April 1982 with a sum of £330 million (to be spent over ten years), and is a joint venture between government and industry. Companies such as Fujitsu, Sharp and Toshiba are taking part in this ambitious project, which intends to leap over the present state of computer technology and create machines far in advance of those that are presently being designed.

The coining of the term 'Fifth Generation' has itself focused attention on the major advances in computer design in the past, and stimulated imaginative possibilities for the future. The first generation of computers were characterised by the use of thermionic valves, but these were made obsolete by the invention of the transistor. Second generation transistor computers were in turn superseded by machines using Large Scale Integration (LSI) technology, which allowed many transistors to be built into a single chip. We are at present at the end of this third generation of computers, but the late 1980's should see the fourth generation of VLSI chips become available. These Very Large Scale Integrated chips will have up to ten million transistors per chip, compared with the current limit of approximately a quarter of a million.

At present, International Business Machines annually spend over £1.1 billion on computer research and development, which makes the Japanese investment seem insignificant in comparison. The Japanese capital outlay is not purely profit-motivated, however.

The focus of science has shifted over the last hundred years from the harnessing of raw energy (in forms as various as electricity and the internal combustion engine) to the study of the most intangible form of wealth — information. Land, labour, capital and industry may have been the source of power in the past, but the future will favour those in control of information. Knowledge and the processing of information will be the keys to post-industrial society. So what is needed for this new society is an engine, a machine with automatic reasoning that can be applied to any factual problem or area of human endeavour with the mathematical precision and certainty of a computer. The engine that is currently being built by the Japanese is called a Knowledge and Information Processing System, or KIPS.

Humans are very good at converting sensory signals into cognitive forms — the state of play in a game of chess can be seen at a glance — but when it comes to taking decisions that depend on large amounts of data we soon discover our limitations. The rules of chess can be explained in a few minutes, yet the game is so complex that grandmasters see only a dozen moves ahead. However, in principle every problem to which reasoning applies can be broken down into a series of simple steps, each of which can be decided by applying rules of inference. This set of rules is known as predicate logic. Logical rules of inference apply to all problems, but for simple everyday decisions we aren't conscious of them.

Logical Language

PROLOG — an abbreviation of Programming Logic — was developed in the early 1970's by the Artificial Intelligence group at the University of Marseilles, though one of its chief developers and proponents is the American Robert Kowalski at Imperial College, London. Based around some of the principles of human logic, it is the language most likely to be used in the Fifth Generation computers. It is also particularly suitable for the creation and interrogation of databases and for educational applications

An expert needs more than a good brain — in the case of a doctor many years of training are required to accumulate medical knowledge. In the same way, a KIPS must have a data bank on which the rules of inference can operate. Furthermore, the system must be extremely user-friendly if the KIPS is not to demand its own breed of experts to operate it. A KIPS machine with which you can hold a conversation in the language of your choice must be a product of research into artificial intelligence — which is an extremely contentious area of study. Thus the targets that the Japanese have set themselves embrace a wide range of computer sciences: hardware, software, interfaces, expert systems (see page 72) and the problems of artificial intelligence.

The Japanese project has been conceived to look beyond advances in chip technology. As the density of transistors in integrated circuits increases, electrons have less distance to travel between each component, and hence the circuits will operate faster. However, the Japanese realise that mere speed is not enough, which is why so much effort is being put into the software. In a

game of chess, for example, there are so many possible sequences of moves (about 10^{120}) that it has been estimated that the time needed to explore all the possibilities exceeds the remaining lifetime of our sun. The project has a target of producing a machine that can make 100 million logical inferences (i.e. can apply 100 million rules) per second. This is referred to as 100 million LIPS (Logical Inferences Per Second).

Another way in which speed could be improved would be by hard-wiring the software functions into the design of the chip, instead of loading them into memory and processing them by means of a general purpose chip. This erosion of the distinction between hardware and software is one of the most interesting aims of the project. There already exist 'associative' memories that have logical search circuits built into the memory cells. These devices can locate a piece of data from the meaning of the data alone — without the need to specify a memory address.

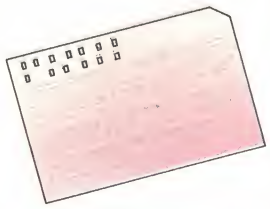
Such advances will speed up the interaction of the logical processors with the data banks. Hard-wiring programming routines into a computer is reminiscent of early computers such as ENIAC (see page 140), but the Fifth Generation machines will diverge from the architecture of von Neumann in one fundamental respect. They will feature many distinct processors all working simultaneously (in parallel), rather than having just one central processing unit. This requires much more care in the timing and control of internal operations but will remove the restriction on speed that the sequential execution of instructions imposes. The internal language chosen for the KIPS is PROLOG, which is a language developed in France and Britain and based on predicate logic. But KIPS will have the ability to communicate in many tongues with its users.

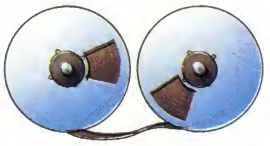
Translation of continuous human speech is another of the project's goals, with an immediate aim of 95 per cent accuracy. At present the ability to recognise even individual words from different speakers lags far behind the manifest success of synthetic speech. However, the NEC Corporation of Japan has already succeeded in creating a machine that can recognise continuous speech. A limitation of this system is that it can recognise the voice of only one individual, and each word must be previously recorded so that the computer can remember and later recognise the speech pattern.

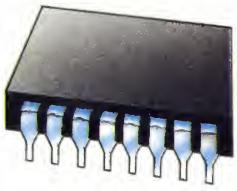
As to the written word: the project is preparing a 100,000 word Japanese/English dictionary and program that it is hoped will permit a translation accuracy of 90 per cent.

Japan has precedents in successful long-term research projects: the PIPS (Pattern Information Processing Systems) project of the 1970's is proving useful in the development of visual data banks and user-friendly interfaces. A KIPS will have to be able to look at an image and extract the salient features and outlines in order to make any preliminary sense of it. On the Tokyo underground there is already a machine that can do this: it scans passageways with a video camera and produces a flow pattern of passengers through the subway system.

Information technology represented an $88 billion business in the USA in 1983, and with employment in the manufacturing industry likely to decline in the same way as that in agriculture did earlier this century (from 40 per cent of the work-force on the land at the turn of the century to three per cent today), the community will move further towards an information society. In the light of this, Japan is doing something very ambitious with its Fifth Generation project. The plan is optimistic and includes a number of 'scheduled' breakthroughs that may or may not materialise (after all, the expected breakthrough in controlled nuclear fusion is still awaited). But it is a positive approach from a trading nation not dissimilar to our own. However, unlike Britain, whose investment in research and development has fallen over the last decade (from 2.32 per cent to 2.09 per cent of the GNP), Japan is speculatively investing in the future.







Generation Game

Round One
The first generation of electronic computers was developed around the technology of the thermionic valve. They had very little in the way of on-line memory, and data was generally stored on punched cards

Round Two
The second generation evolved out of the transistor, which increased the memory capacity, though off-line storage (in the form of magnetic tape) was still used

Round Three
The invention of the integrated circuit increased computer power dramatically, and was ultimately responsible for the microcomputer — characterised by the floppy disk drive

Round Four
We are now moving from the third to the fourth generation, which will be based on VLSI chip technology. The RAM memory will be so large that off-line storage will become less important

Round Five
The Fifth Generation of computers, being developed primarily in Japan, is really concerned with software rather than hardware. However, it is based on the assumption that user memory will be so large that program size will cease to be a consideration

KEVIN JONES

Research Machines 380Z

A robust physical construction and superb high resolution graphics have made this microcomputer very popular in schools and with the military

The products of Research Machines Limited represent one of the most enduring classes of computer available. Although these machines are not particularly innovative, nor especially competitively priced, they are extremely solidly designed and constructed, well-supported and extraordinarily reliable. The company's most popular computer, the RML 380Z, won't be found in many homes but, as one of the most common educational machines, it represents for many children their first experience of computing.

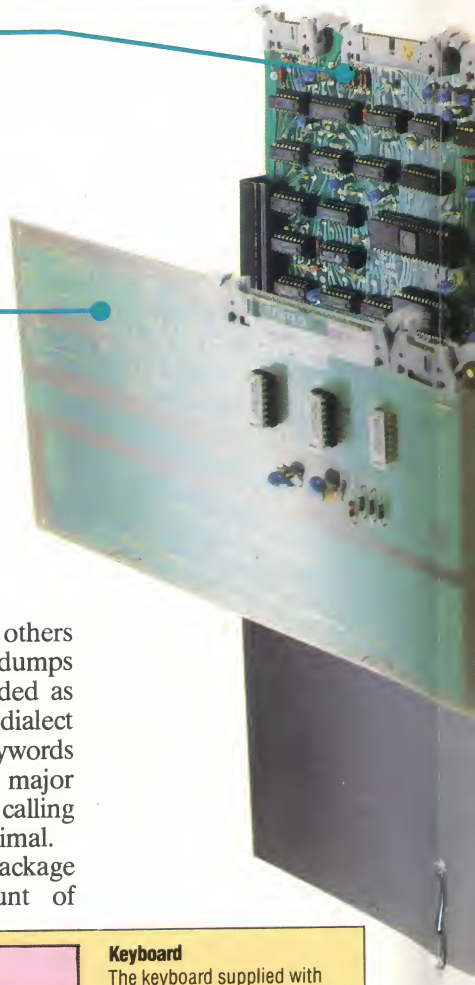
Compared with most machines the 380Z is huge: the main circuit boards are kept in a robust 19 inch (48cm) wide casing, complete with handles at the sides. Removing the lid of this 'black box' reveals why it's so big, as nearly a quarter of the space inside is taken up by the power supply. Though sheathed in metal, its weight makes it clear that this is not an advanced switching-type power supply unit, but a solid iron-cored transformer with enormous capacitators. This may seem old-fashioned, but it has the considerable advantage of being almost impossible to overload or otherwise damage.

It is perhaps this reliability that has made the 380Z popular at the Ministry of Defence, where a large number are used for stock control and similar chores. In those schools and colleges that offer higher-level maths, physics and science, the machine is particularly favoured for its high resolution graphics, which prove useful for pictorial demonstrations of various aspects of the curricula.

The High Resolution Graphics (HRG) package is a set of machine code routines that are called from the user's program, and which can alter the display generated by the HRG card. This must be present in order to produce graphics at all; and although it was originally introduced some years ago, it remains one of the better systems available. By altering the contents of certain memory locations the card can generate displays at several resolutions in the normal colours (red, yellow, green, blue, magenta, cyan). Depending on the resolution chosen, which can range from 160 x 96 to 320 x 192, these six colours plus white can be given up to 255 different levels of brightness, thus increasing the range of colours to 1,786 (seven times 255, plus black). Alternatively, it is possible to use some of the bits normally intended for specifying intensity to produce multiple pages of graphics, though the number of different intensities will be correspondingly lower.

Monitor
To take full advantage of the 380Z's facilities, a colour monitor, with an RGB interface, is essential. The machine can be purchased with 40 or 80 screen columns as standard, and this determines the type of monitor needed

CHRIS STEVENS



Disk Drive Controller Board
As well as carrying a specialised disk controller chip, this board has a Z80 clock/timer chip (CTC), and an 8521 serial Input/Output chip, which together provide impressive communications facilities

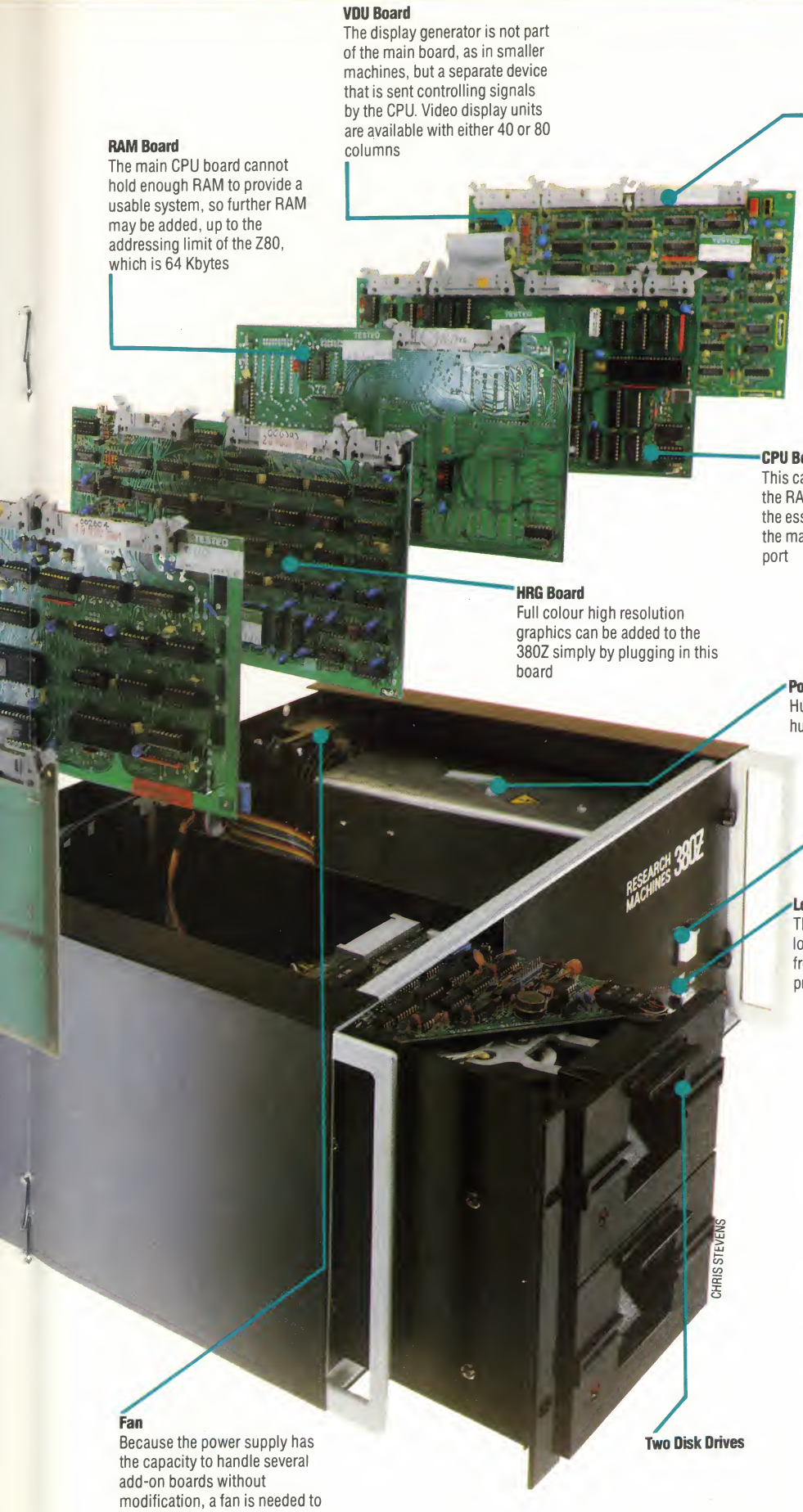
Bus Terminator Board
This is situated at the far end of the bus to the CPU board, and guards against interference on the various electrical lines

A full set of these calls, plus a number of others for handling such related matters as printer dumps (copying the screen to paper), are provided as extended versions of the RML BASIC. This dialect is much like Microsoft BASIC, and most keywords are used in an identical fashion. The only major exception is the use of text labels for calling subroutines, rather than an address in decimal.

However, the interpreter plus HRG package together occupy a considerable amount of

Keyboard
The keyboard supplied with the RML 380Z is mounted in a small but weighty metal box. The keys are arranged in a fairly standard pattern, and are of high quality, with a solid but pleasantly light touch. They are obviously designed to withstand heavy use, a factor that makes the keyboard an ideal choice for the classroom

CHRIS STEVENS



VDU Board
The display generator is not part of the main board, as in smaller machines, but a separate device that is sent controlling signals by the CPU. Video display units are available with either 40 or 80 columns

RAM Board
The main CPU board cannot hold enough RAM to provide a usable system, so further RAM may be added, up to the addressing limit of the Z80, which is 64 Kbytes

Z80 Bus Connections

CPU Board
This carries the Z80 CPU, part of the RAM, the ROM, and most of the essential parts needed to run the machine, such as a keyboard port

HRG Board
Full colour high resolution graphics can be added to the 380Z simply by plugging in this board

Power Supply
Huge, heavy and very hard to hurt

Reset Button

Lock
This allows the machine to be locked on to prevent anyone from interfering with critical programs

Fan
Because the power supply has the capacity to handle several add-on boards without modification, a fan is needed to assist cooling

Two Disk Drives

Research Machines 380Z

PRICE
£2,062 (5in disk system)
£3,395 (8in disk system)
Educational discounts available

SIZE
595 x 425 x 215mm

CPU
Z80

CLOCK SPEED
4 MHz

MEMORY
Up to 6 Kbytes ROM
56 Kbytes RAM

VIDEO DISPLAY
24 lines of 40 or 80 characters, seven colours with up to 255 shades. Graphics resolution of 320 x 192 and 160 x 96

INTERFACES
RS232 serial, cassette, parallel printer

LANGUAGE SUPPLIED
Research Machines extended BASIC

OTHER LANGUAGES AVAILABLE
ALGOL, FORTRAN, and CP/M standards

COMES WITH
Manuals for installation, CP/M, disk system, cassette system and BASIC utility programs on disk

KEYBOARD
60 word-processor quality keys

DOCUMENTATION
Excellent, though a little dry. The information is comprehensive and easily referenced

memory, and may not leave sufficient room for sophisticated user programs. For this reason, three versions of the BASIC are provided, with either all, some or none of the HRG package included, depending on how constraining your memory requirements are.

The use of the HRG package is not limited to BASIC, and as a simple machine code file it can be accessed from any language. Since the 380Z is generally run under CP/M (see page 410), this means that a wide range of languages are available, but the machine is highly unusual in the microcomputer world in having a version of ALGOL configured to run on it.

This language, which many European scientists prefer to FORTRAN (the language favoured in North America for science purposes), resembles PASCAL and is particularly strong in complex mathematical calculations, such as those involved in structural design. This is another factor that makes the machine attractive to educationists.

Sounds Ideal

The Commodore 64's Basic doesn't match up to its remarkable sound facilities

Among the current range of home computers, the Commodore 64 is supplied with the most sophisticated sound-making facilities. These are attributable to a specialised chip called the Sound Interface Device — or SID, as it is better known.

SID provides capabilities similar to that of a commercial monophonic synthesiser. There are three oscillators with an eight-octave range (0-3900Hz in 65,536 steps); a master volume control, from 0 to 15; four waveforms for each oscillator (triangle, sawtooth, variable pulse and noise); oscillator synchronisation; and envelope generators allowing ADSR control for each oscillator. Further features include: ring modulation; programmable filter with low pass, band pass, high pass, notch output (which blocks out a narrow band of frequencies) and variable resonance; envelope filtering; two analogue-to-digital potentiometer interfaces that can be used to control SID facilities; and an external audio input, which enables additional SID chips to be linked together. Other audio signals can be input, filtered and mixed with the standard SID outputs.

It would be impossible to detail the operation of each of these features here (several good books are available), but we can explain what all these phrases mean. First of all, oscillator synchronisation causes two signals (in this case two specified voices) to be harmonically locked together, making a single, more complex tone out of the two separate signals.

Modulation is the modification of one signal by another, affecting either the frequency or amplitude (volume) of the sound. Ring modulation is the amplitude modulation of one voice by another. This results in a tone that is clear but has a jarring, discordant effect and can be used to produce bell-like sounds similar to those of steel drums. Such sounds are said to have inharmonic overtones.

Filters enable specified frequency ranges to be eliminated from a signal. The different types of filtering possible on the Commodore 64 have effects that are suggested by their names: low pass filters cut out frequencies higher than a specified frequency; band pass filters eliminate frequencies above and below a specified 'band' of frequencies; notch filters are the inverse of band pass filters — they cut out a specified band; high pass filters cut

Guiding Light

Player-Missile graphics are one of the strong points of the Atari machines

Player-Missile Graphics

Player-Missile or 'PM' graphics form an important part of the Atari's graphics capabilities. They are similar in nature to the sprite graphics available on the Commodore 64 (see page 408) and the Sord M5, allowing the programmer to design and control up to eight different high resolution shapes. These movable shapes operate independently of any background display and may be programmed to move either in front of or behind any other shapes drawn on the screen. This allows the programmer to add a third dimension to the screen effects. PM graphics can be moved smoothly, at speed, across the screen and so are ideal for fast-moving arcade games. They can also be used to create more colourful static displays than are possible using the normal graphics modes, as PM objects can be coloured independently of each other and of the background display.

As with all sprite graphics, the secret of PM graphics' facilities lies in dedicated hardware. Special registers are designed to control the movement, colour and screen display of the PM objects. All the programmer has to do is place certain values in these registers to manipulate the objects. In BASIC this is done using the POKE command. Once a number is POKEd into the relevant register then the Atari's own hardware takes over the rest of the work. This is done at machine code speed and is therefore much faster than if the process was controlled from BASIC.

Let us now look at the creation of PM objects and the registers that control them. Players are designed from a vertical strip, eight pixels wide and 128 or 256 pixels high. Each row across the strip is represented as a single byte in the computer's memory. By POKEing suitable binary codes it is possible to define the shape of a player using a similar method to that used to create user-defined characters (see page 246). Up to four players may be defined in this way, each taking up

```
10 SID=54272
20 POKESID+23,0
30 POKESID+24,15
40 POKESID+5,40
50 POKESID+6,201
60 FOR N=1TO5
70 READ FH,FL,D
80 POKESID+1,FH:
   POKESID,FL:
   REM*PLAY
   NOTE*
90 POKESID+4,33
100 FORI=1TO300*
   D:NEXT I
110 POKESID+4,32
120 FORI=1TO100:
   NEXT I
130 NEXT N
140 FORI=1TO2000:
   NEXT I
150 POKESID+24,0
160 REM**FH FL D**
170 DATA 57,172,1
180 DATA 64,188,1
190 DATA 51,97,1
200 DATA 25,177,1
210 DATA 38,126,2
220 END
```

out frequencies lower than a specified one; and variable resonance can be applied to all the above filters to emphasise the frequencies around the cut-off points. Envelope filtering is a special case: it has a different effect from the others in that digitised ADSR values set for envelope 3 can be read from the SID chip and applied to a signal in such a way that the harmonic structure changes throughout the course of a note. It works like a variable filter.

These sophisticated features enable you to build highly complex sounds into interesting effects and convincing emulations of conventional instruments. The daunting aspect of SID is that CBM BASIC V2, the dialect supplied with the 64, provides no commands dedicated to sound at all. Control is exercised by PEEKing from and POKEing into the 29 SID control registers. A lot of BASIC code is therefore needed to generate even simple effects, and in some cases BASIC isn't fast enough to do full justice to the full range of SID's possibilities.

A full description of the SID control registers would require more space than a complete issue of THE HOME COMPUTER COURSE, but it is possible to play notes with pleasing tones as shown in the program on the left.

Although the program is 22 lines long, it plays merely five notes of a simple tune on one oscillator. Line 20 disconnects the filter from the oscillators; line 30 sets the master volume at its maximum; and lines 40 and 50 specify a piano-like envelope. Line 80 sets the note frequency; 90 and 100 start and stop the ADSR cycle and select a sawtooth wave for voice 1; and timing is achieved with FOR . . . NEXT loops in lines 100, 120 and 140.

Programming sound on the Commodore 64 in BASIC is a major effort in terms of both learning and writing code. Moreover, it can be a very frustrating exercise, as the only way to discover if a more complex set of BASIC statements will run in an acceptable time is by trial and error. If you want simpler methods of sound generation it is worth investigating the many sound editing programs that are commercially available. These are usually written in machine code and make the most of the marvellous features of the Commodore 64.

its own 256 or 128 bytes of memory.

Each of the four players has a missile figure associated with it that is two bits wide. To create players and missiles it is necessary to POKE the bit patterns that define their shape into a certain area of memory. The area of RAM used can be chosen by the programmer but the computer must be informed by setting a pointer to the beginning of the area.

If the programmer elects to use single-pixel vertical resolution then twice as much memory is required than for two-pixel vertical resolution. The following program designs player 0 in two-pixel vertical resolution as a space ship:

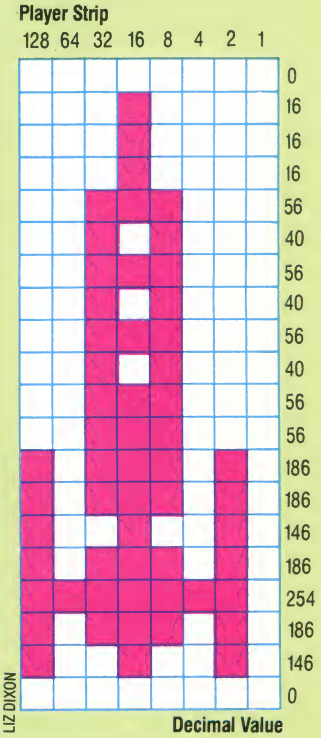
```
10 REM *** DEFINE A PLAYER ***
20 P=PEEK(106) -8:REM SETS P TO 2K BELOW
   TOP OF RAM
30 POKE 54279,P:REM SETS POINTER TO PM
   AREA
40 BASE = 256*P:REM SETS PM AREA BASE
   ADDRESS
50 FOR I = BASE+512 TO BASE+640
60 POKE I,0:REM CLEAR PLAYER 0 AREA
70 NEXT I
80 FOR I = BASE+512+50 TO BASE+530+50
90 READ A:POKE I,A:REM DEFINE FIGURE
100 NEXT I
110 DATA 16,16,16,56,40,56,40,56,40
120 DATA 56,56,186,186,146,186,254,186,146
```

Each player figure has several registers associated with it. These registers control colour, horizontal position and size. The last of these enables the programmer to increase the width of a player by a factor of two or four. Further registers control player-to-background priority. Missiles take on the colour of their parent player but missile size can be changed independently. For games applications a series of registers is set aside to detect on-screen collisions between players, missiles and background. However, there is no vertical position register for missiles or players. Vertical movement must be achieved by moving the contents of each location that holds the bit patterns for the figure up through the area of memory set aside for that player. This is a fairly straightforward task in Assembly language but would be relatively slow in BASIC. It is a good idea to try to make characters that move vertically as short and stubby as possible.

Player-Missile graphics considerably extend the Atari's graphics potential, although they are not as versatile or as easy to use as the Commodore 64's sprites. Here is a continuation of the program started earlier, to colour the space ship and move it from left to right across the screen.

```
130 POKE 559,46:REM ENABLE PM 2 LINE
   DISPLAY
140 POKE 53277,3:REM ENABLE PM DISPLAY
150 POKE 704,88:REM COLOUR PLAYER 0 PINK
160 GRAPHICS 0
170 SETCOLOUR 2,8,2:REM SET BACKGROUND TO
   DARK BLUE
180 FOR I = 0 TO 320
190 POKE 53248,I:REM SET HORIZONTAL
   POSITION
200 NEXT I
210 END
```

Rocket PM
Before a player object can be defined, it must first be drawn out and the decimal values for each row of pixels calculated



Language Lab

To conclude our course, we take a critical look at the Basic language, and at some of the alternatives to it

As a postscript to our Basic Programming course, we would like to discuss briefly some of the strengths and weaknesses of BASIC compared with other programming languages.

BASIC is an off-shoot of FORTRAN, one of the earliest programming languages. Unlike most other languages, BASIC is interpreted. This means that when a BASIC program is executed, a special program elsewhere in the computer's memory interprets the code line by line and converts the BASIC statements into machine code. Here's what would happen in a short BASIC program like this:

```
10 CLS
20 PRINT "TYPE IN A NUMBER"
30 INPUT X
40 PRINT "TYPE IN A SECOND NUMBER"
50 INPUT Y
60 "PRINT "THE PRODUCT OF THE TWO NUMBERS
   IS: ";
70 PRINT X*Y
80 PRINT
90 PRINT "DO YOU WANT ANOTHER GO?"
100 PRINT "PRESS 'Y' TO TRY AGAIN"
110 PRINT "OR 'N' TO END"
120 FOR X = 1 TO 1
130 LET A$ = INKEY$
140 IF A$ < > "Y" AND A$ < >"N" THEN X = 0
150 NEXT X
160 IF A$ = "Y" THEN GOTO 10
170 END
```

When the BASIC interpreter encountered line 10 it would work out the machine code needed to clear the screen. For line 20 it would work out the machine code instructions necessary for sending the TYPE IN A NUMBER message to the screen. For line 30 it would set up the memory space needed to store a real number, wait for input from the keyboard and then convert the number typed in into binary and store it in the space allocated to variable X. All this would be repeated for lines 40 to 60. If the user wanted to repeat the program by typing Y, the interpreter would branch back to line 10 and repeat all the calculations and computations again.

Most languages other than BASIC are 'compiled'. This means that after the program has been written it is processed by a 'compiler' before it can be run. The compiler is a separate program that goes through the 'source code' (the original program) and produces a second version of it in machine code. When the compiled program is run, it is likely to work very much faster than an interpreted program because all the time-consuming translations into machine code have already been done.

If compiled programs work so much faster than interpreted programs, you might wonder why all programming languages don't use compilers. There are several advantages to using interpreted programs, such as BASIC. Most of these stem from the fact that it is an interactive language, which is one that can be tested and de–bugged 'at the keyboard' while the program is being developed. BASIC, for example, allows the STOP command to be inserted at any point in the program. When the interpreter encounters a STOP statement, it stops interpreting the program and allows 'commands' to be issued from the keyboard.

Commands are instructions that can be directly executed by the interpreter when the program is not running. BASIC is provided with a large number of these and they can be invaluable in de-bugging. After a BASIC program has been executed (i.e. the interpreter has encountered the END statement) or when the interpreter encounters a STOP statement, it is possible to PRINT the values of all the variables. Try running the address book program, for example. Run the program and type 9 to exit from the program. If it runs all the way through without any error messages appearing, it should end with the BASIC prompt (this is usually an OK, > or *). Then type PRINT RMOD<CR>. The interpreter should print a 0 on the screen (provided you have not added any records!). Then try PRINT SIZE<CR>. The interpreter will print a number on the screen one larger than the number of records you have in the data file.

Basic's Advantages

BASIC is often called the ideal language for the inexperienced programmer because it allows bugs to be removed at the keyboard. It has another great advantage: it is comparatively easy to learn. For example, in the Basic Programming course, we have covered all the fundamentals and many of the advanced aspects of BASIC in just 86 pages. Syntax errors such as 40 PRNT A(12) will usually result in easily understandable error messages when the program is executed, such as SYNTAX ERROR IN 40. A glance at the line number referred to usually makes it clear where the error lies, and rectifying the error is usually no more difficult than typing EDIT 40<CR> (followed by a

few editing commands) or re-typing the correct line. Whenever the BASIC interpreter encounters an error in syntax or logic it stops the execution of the program and reports the error. Fixing the bugs is as simple as trying a new line in place of the erroneous line, and typing RUN<CR> again.

Basic's Disadvantages

The BASIC programming language has a number of disadvantages, however, some of which are subtle and some glaring. Because it is interpreted (although a few compiled versions of BASIC do exist) it runs very slowly. If speed is not very critical (as in a program to calculate your current bank balance, for example) the slowness of interpreted BASIC will be of no consequence. If, on the other hand, speed is of the essence (as in a screen animation program using graphics, or a 'clock' used to time reactions in a laboratory experiment) interpreted BASIC is likely to be far too slow.

If you need speed in your programs, there are two routes to follow: programming in either machine code or Assembly language (see page 448) — a difficult and time consuming process — or programming in a compiled language such as PASCAL or FORTH. Compiled languages are not difficult to learn, but the source code (the original program) is almost sure to contain bugs, which the compiler will find when it tries to compile the program. These are difficult to rectify, compared with bugs in BASIC. After corrections have been made to the source code, the program will have to be compiled all over again. Most compilers take two or three 'passes' through the source code, and each pass is likely to result in error messages, each of which will have to be corrected before the program can be re-compiled.

Producing a correctly compiled program is likely to be a far more time consuming processs than achieving a working program in interpreted BASIC. On the other hand, BASIC is likely to lead the novice programmer from the 'straight and narrow' by allowing bad programming techniques that highly structured languages such as PASCAL would reject. BASIC allows the programmer to write very careless programs, full of GOTOs for example, and these bad habits can make the transition to more advanced languages difficult.

What Next After Basic?

BASIC is a flexible language, and one that is not difficult to learn. It has excellent string handling facilities, but is slow and fails to take full advantage of the power of a home computer. On the other hand, more modern languages, such as PASCAL and FORTH, offer programming facilities either difficult or impossible in BASIC.

PASCAL was also devised as a teaching language, and specifically designed to encourage the development of well constructed, 'structured' programs. PASCAL is a compiled language, which means that users encounter numerous errors picked up by the compiler (after the source code has been written and before the compiled 'object code' can be run), and this can be very frustrating. Novice PASCAL programmers also tend to find the restraints of the language, such as the need to declare all variables at the beginning of the program (and to state what type they are — real, integer, etc.), to be an impediment to free and flexible programming.

On the other hand, PASCAL demands that the programmer thinks through the logic of the program properly before writing. Programs in PASCAL are likely to throw up numerous syntactical errors in the source code. But they are also more likely to be well designed and less likely to contain fundamental logical errors.

FORTH has recently become a very popular alternative to BASIC as a programming language on home micros. Although FORTH is not as difficult to learn as Assembly or machine code language, it must be said that it is far less 'intuitive' than either BASIC or PASCAL. Even so, FORTH has many unique merits that make it a contender as the programmer's second language.

Although FORTH is a high level language, it runs nearly as fast as machine code, owing to the unique way it works. Whereas languages such as BASIC have a fixed number of statements and commands, FORTH users can define their own vocabulary.

The keyword PRINT in BASIC means that any character following it enclosed in double quotes will be printed on the screen. Nothing the programmer does can alter this. In FORTH, PRINT could be defined to produce, say, a listing on the screen of the hexadecimal equivalents of the ASCII codes, printed in a vertical column, of the characters in a string.

FORTH gives the programmer the power to define any word to mean whatever is wanted and to produce the desired results whenever it is used from then on. Not only is FORTH extremely flexible in this way, but it also produces programs that can be compiled to object code (see page 184) which are nearly as compact and fast running as machine language programs.

Although there are many programming languages available, most hobbyists moving on from BASIC will be inclined to choose from Assembly language, PASCAL and FORTH. Very briefly, the advantages and disadvantages of each can be summarised thus:

BASIC
Easy to learn
Easy to remember
Easy to de-bug
Slow in execution
Uses lots of memory
Does not encourage structured programming

Assembly Language:
Not very easy to learn
Not very easy to remember

Difficult to de-bug
Very fast in execution
Gives complete control over the microprocessor

PASCAL
Moderately easy to learn
Moderately easy to remember
De-bugging more difficult than in BASIC
Encourages better programming techniques
Execution faster than BASIC but slower than Assembly
Needs to be compiled, which takes time; once correctly compiled, runs nearly as fast as Assembly
Gives fair control over the microprocessor, but less than Assembly; string handling not as easy as in BASIC

FORTH
Not very easy to learn; easier for complete beginners, not so easy for BASIC programmers
Moderately easy to remember
De–bugging in interpreter mode very easy
Can be compiled; executes almost as quickly as Assembly language
Gives complete control over the microprocessor
Very economical on memory
Easier to learn than Assembly language, though less 'intuitive' than BASIC

Basic Flavours

LYNX 96

```
1  REM *CREATE DATA FILE*
2  DIM N$(30)
3  LET N$="@FIRST"
4  DIM F$(15)
5  LET F$="DUMMY"
6  LET Z=2
7  EXTBACK 1
8  EXTSTORE 1,Z,N$,N$,N$
9  EXTSTORE 1,F$,F$,F$,F$
10 INPUT "INSERT DATA TAPE, PRESS
   RECORD, & TYPE 'Y'";A$
11 SSAVE 1, "ADDBKDAT"
12 PRINT "STOP THE TAPE, AND REWIND"
13 END
```

NB This is the initialising program for the 96K Lynx; we have no information on cassette file handling for the other models.

Main Program Variables
Copy the Spectrum list with these substitutions for the numeric variables:

Replace: SIZE by Z
RMOD by R
SRTD by D
CURR by C
CHOI by H
BTM by b
MD by m
TP by t

and make the following line changes, substitutions, and deletions:

```
1100 REM *CREARR* S/R
1110 DIM N$(30)(50)
1120 DIM M$(30)(50)
1130 DIM S$(30)(50)
1140 DIM T$(15)(50)
1150 DIM C$(15)(50)
1160 DIM R$(15)(50)
1170 DIM X$(15)(50)
1180 DIM Z$(30)
.
1210 LET Z=0
1220 LET R=0
1230 LET D=1
1240 LET C=0
1250 LET Z$="@FIRST"
1260 LET Q$=""
1300 RETURN
.
1400 REM *RDINFL* S/R
1405 PRINT "INSERT DATA TAPE AND PRESS
     PLAY"
1410 GOSUB 3100
1420 SLOAD 1, "ADDBKDAT"
1430 PRINT "STOP THE TAPE"
1440 GOSUB 3100
1450 EXTBACK 1
1460 EXTFETCH 1,Z
1470 FOR K=1 TO Z-1
1480 EXTFETCH 1,
     N$(K),M$(K),S$(K),T$(K),C$(K),R$(K)-
     ,X$(K)
1490 NEXT K
1500 LET Q$=N$(1)
1510 RETURN
.
3120 IF KEY<>32 THEN LET L=0
.
3780 LET A$=KEY$
.
3810 LET H=VAL(A$)
3820 IF (H<1) OR (H>9) THEN LET L=0
.
4500 REM *MODNAM* S/R
4510 REM CONVERT TO U/CASE
4520 LET D$=UPC$(N$(Z))
     (delete lines 4530-4590)
.
4600 LET P$=""
4601 LET A$=""
4602 LET T=LEN(D$)
4603 LET S=0
.
4610 REM LOCATE LAST SPACE
.
4630 IF MID$(D$,L,1)=" " THEN LET S=L
.
4670 IF MID$(D$,L,1)>"@" THEN LET
     P$=P$+MID$(D$,L,1)
.
4710 IF MID$(D$,L,1)>"@" THEN LET
     A$=A$+MID$(D$,L,1)
```

Lines 5410 to 5460 must be reduced to single statements, for example:

```
5410 LET U$=N$(L):LET N$(L)=N$(T): LET
     N$(T)=U$     becomes
5410 LET U$=N$(L)
5411 LET N$(L)=N$(T)
5412 LET N$(T)=U$
```

and so on, no changes otherwise.

```
.
5600 REM *SAVREC* S/R
5605 PRINT "INSERT DATA TAPE AND PRESS
     RECORD"
5610 GOSUB 3100
5620 EXTBACK 1
5630 EXTSTORE 1,Z
5640 FOR K=1 TO Z-1
5650 EXTSTORE 1,
     N$(K),M$(K),S$(K),T$(K),C$(K),R$(K)-
     ,X$(K)
5660 SSAVE 1, "ADDBKDAT"
5670 PRINT "STOP THE TAPE"
5680 GOSUB 3100
5690 RETURN
.
5855 LET X=0
```

```
5860 LET m=INT((b+t)/2)
5870 IF M$(m)=U$ THEN LET X=1
5880 IF U$>M$(m) THEN LET b=m+1
.
6080 LET A$=KEY$
.
6110 IF A$=" " THEN RETURN
6120 GOSUB 6200
6130 RETURN
.
6730 FOR I=1 TO 1
6735 LET I=0
6740 LET A$=KEY$
6750 IF (A$=E$) OR (A$=" ") THEN LET I=1
6760 NEXT I
```

This fragment must be reproduced at lines 6880-6910, 6990-7030, 7110-7140, 7220-7250, 7640-7670

Initialising Program

This is the initialising program for the Dragon 32.

DRAGON 32

```
1 REM *CREATE DATA FILE*
2 LET Z=2
3 LET N$="@FIRST"
4 OPEN "O", #-1, "ADBKDAT"
5 INPUT "INSERT DATA TAPE, PRESS
  RECORD, & TYPE 'Y'";A$
6 PRINT#-1,Z,N$,N$,N$,N$,N$,N$,N$
7 CLOSE #-1
8 PRINT "STOP THE TAPE, AND REWIND"
9 STOP
```

BBC MICRO

On the BBC Micro replace lines 4, 6 and 7 by:

```
4 F1=OPENOUT("ADBKDAT")
6 PRINT #F1,Z,N$,N$,N$,N$,N$,N$,N$
7 CLOSE#F1
```

COMMODORE 64

On the Commodore 64 and Vic-20 replace lines 4, 6 and 7 by:

```
4  OPEN 1,1,2,"ADBKDAT"
6  PRINT #1,Z:PRINT #1,N$:PRINT #1,N$:
   PRINT #1,N$:PRINT #1,N$:PRINT #1,N$:
   PRINT #1,N$:PRINT#1,N$
7  CLOSE 1
```

VIC-20

Main Program Variables

On the Dragon, the Commodores, and the BBC Micro copy the Spectrum list (published in full on page 458) with these substitutions for the numeric variables.

Replace: SIZE by Z
RMOD by R
SRTD by D
CURR by C
CHOI by H
BTM by BT
MD by MD
TP by TP

and make the following line changes, substitutions, and deletions:

```
1100 REM *CREARR* S/R
1110 DIM N$(50)
1120 DIM M$(50)
1130 DIM S$(50)
1140 DIM T$(50)
1150 DIM C$(50)
1160 DIM R$(50)
1170 DIM X$(50)
```

Delete lines 1180-1190

```
1210 LET Z=0
1220 LET R=0
1230 LET D=1
1240 LET C=0
1250 LET Z$="@FIRST"
1260 LET Q$=""
1300 RETURN
```

This is the Dragon 32 version of subroutine 1400:

```
1400 REM *RDINFL* S/R
1410 OPEN "I",#-1, "ADBKDAT"
1420 PRINT "INSERT DATA TAPE AND PRESS
     PLAY"
1430 GOSUB 3100
1440 INPUT#-1,Z
1450 FOR K=1 TO Z-1
1460 INPUT#-1,N$(K),M$(K),S$(K),T$(K),-
     C$(K),R$(K),X$(K)
1470 NEXT K
1480 Q$=N$(1)
1490 CLOSE#-1
1500 PRINT "STOP THE TAPE"
1510 GOSUB 3100
1520 RETURN
```

In the preceding list, on the BBC Micro replace line 1410 by:

```
1410 F1=OPENIN("ADBKDAT")
```

and replace #-1 by #F1 in lines 1440, 1460, 1490

In the preceding list, on the Commodore 64 and Vic-20 replace line 1410 by:

```
1410 OPEN 1,1,0,"ADBKDAT"
```

Replace #-1 by #1 in lines 1440 and 1460, and replace 1490 by:

```
1490 CLOSE 1
```

On the BBC Micro replace INKEY$ by INKEY$(0) throughout; and replace INPUT"..message..";A$ by INPUT"..message..",A$. On the Commodores replace LET A$=INKEY$ by GET A$ throughout; and replace IF INKEY$... by GET G$: IF G$...

On the BBC Micro, the Dragon, and the Commodores in subroutine 4500 replace all references to D$(L) by MID$(D$,L,1). Replace CODE... by ASC(...). Replace FIRST by LAST in line 4610.
Delete : LET L=T from the end of line 4630.

This is the Dragon version of subroutine 5600; for BBC and Commodore variations see the Initialising Program notes above.

```
5600 REM *SAVREC* S/R
5610 OPEN "O",#-1, "ADBKDAT"
5620 PRINT "INSERT DATA TAPE AND PRESS
     RECORD"
5630 GOSUB 3100
5640 PRINT#-1,Z
5650 FOR K=1 TO Z-1
5660 PRINT#-1,Z,N$(K),M$(K),S$(K),T$(K)-
     ,C$(K),R$(K),X$(K)
5670 NEXT K
5680 CLOSE #-1
5690 PRINT "STOP THE TAPE"
5693 GOSUB 3100
5695 RETURN
```

In subroutine 6200 on the BBC Micro insert:

```
6205 VDU 2
6275 VDU 3
```

and replace LPRINT by PRINT.

On the Commodores insert:

```
6205 OPEN 4,4:CMD 4
6275 PRINT#4:CLOSE 4
```

and replace LPRINT by PRINT.

On the Dragon replace LPRINT by PRINT#-2.

Firm Foundations

In the history of the microcomputer, developments in hardware and software are inextricably linked, and it is as much a story about personalities as products

There have been several episodes in history in which the pace of technological change has left people bewildered. But nothing to date — not even the progress of flight, from the Wright brothers to lunar exploration — can match the speed of the microelectronics revolution. The progress from the first primitive microprocessors to today's 16-bit designs, from the first microsystems to today's desktop mainframes, has taken just a decade. And the speed of development is still increasing.

Around 1971, several of the new chipmaking firms in California concluded that the main functions of a computer could be housed on a single sliver of silicon. There were no grandiose plans for a revolution then, and no talk of 'information technology'. The idea was to produce a small and cheap computer that might be used to control factory machines or lifts, and the first microprocessors were well suited to such tasks.

One of these chipmakers, Intel, is generally credited with producing the first microprocessor, called the 4004. The 'fours' in the number refer to its power: it was a four-bit processor handling data in blocks of four binary digits. It could only use small quantities of memory — just enough for a lift control program, for example.

By 1972 Intel had developed the 8008 chip, an eight-bit processor, and hobbyists began to think about building computers for themselves around the new chip. Articles in American hobby electronics magazines described how to do it, and although the resulting computers did not have monitor screens, proper keyboards or other sophisticated aids, they were the first home machines. It was from one of these hobby projects that what could be called the first commercial home microcomputer, the Altair 8800, evolved. This was available only in kit form, however.

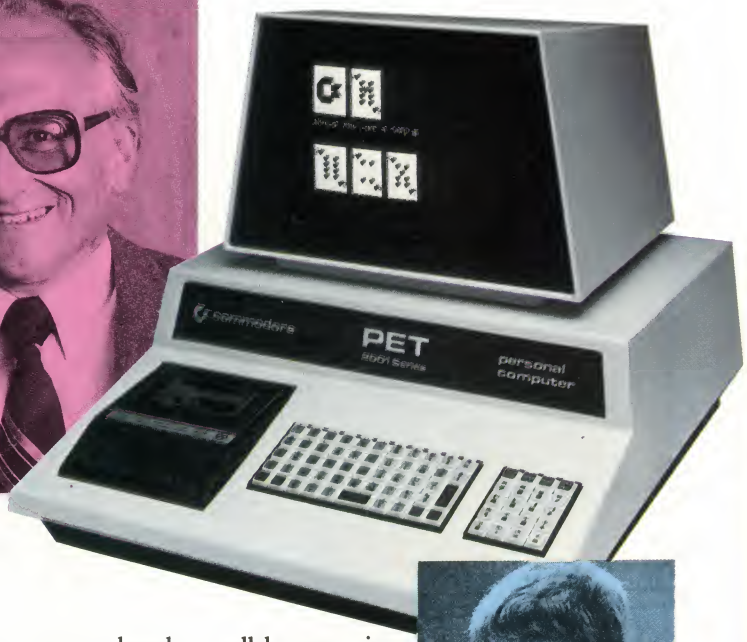
Then in the next year, came the first 'real' microprocessor, the 8080, again from Intel. This operated on eight-bit blocks of data and could handle up to 64 Kbytes of memory for bigger programs. By this time the other chip firms were starting to catch up. Motorola's 6800 chip did much the same as the 8080. It had similar hardware characteristics but required different instructions to make it work. This is the point where software compatibility problems started: programs written for the 8080 would not run on the 6800, and vice versa.

At the same time, other firms had developed similar processors, among them National Semiconductor, Signetics and Advanced Micro Devices. But the next major prime mover was MOS Technology, where one of the leading characters in our story, Chuck Peddle (see page 180), was working. Peddle was at MOS Technology when the company developed a processor very like Motorola's 6800, called the 6500. In fact, it was so close to the 6800 that changes had to be made and the revised chip was eventually given the name 6502.

Founding Fathers
Though Chuck Peddle designed both the Commodore PET and the 6502 microprocessor that it was based around, the contribution of Bill Gates, as the author of the Microsoft BASIC built into the PET's ROM, was equally important

COURTESY OF COMMODORE

Chuck Peddle



Commodore was already well-known in Canada for office machinery and electronic calculators. Peddle joined the firm with the idea of developing a personal computer complete with screen, keyboard, cassettes for program storage, and everything else a real computer should have — all built, of course, around the 6502 processor. The machine emerged in 1976 as the PET 2001, a friendly name chosen to convey the idea that the computer was not too advanced for the home user.

But as the first PET was becoming available, two more innovators were preparing to market a computer from a Californian garage. Steve Wozniak (see page 155) had always wanted to own a computer, and joining the Homebrew Computer Club showed him it could be done. He designed a computer on a single circuit board, and with his friend Steve Jobs began making and

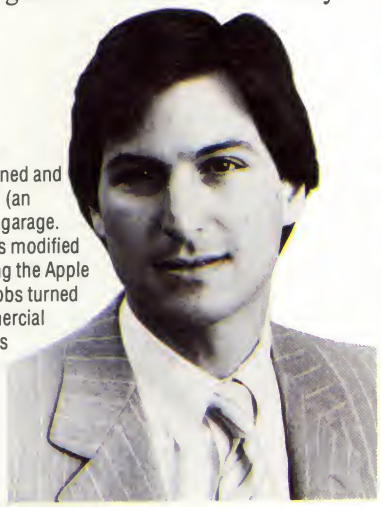
TONY SLEEP

Bill Gates

selling these. They called their board the Apple I. Housed in a box with a keyboard, the machine eventually transformed itself into the enormously successful Apple II. This machine emerged just after the Peddle PET and spawned a cottage industry of software and hardware manufacture.

The Tandy Corporation of Fort Worth, Texas, had ideas of its own for the small computer market. The corporation was, and remains, a manufacturer of a wide variety of electrical goods such as hi-fi, synthesisers and radios, selling them through its chain of stores. The home computer represented a natural extension of its range, and in the Radio Shack shops it already had a distribution network across the US. The result was the TRS-80 Model 1, another huge success in the US market. TRS simply stands for Tandy Radio Shack, but the 80 refers to the microprocessor used, the Zilog Z80. Zilog was yet another new chip firm, and had produced a processor similar to the Intel 8080 but with substantial improvements.

With the TRS-80 Model 1 having a Z80 microprocessor, and the Apple II and Commodore PET having 6502s, home computers began to exhibit a diversity in hardware. But along with this first major consumer choice came the associated problems of machine incompatibility and non-standard software. The kind of microprocessor used in the early machines is significant because the chip determines the choice of software that becomes available from third parties. While the hardware was being developed, standards in software were being set as well.



COURTESY OF APPLE

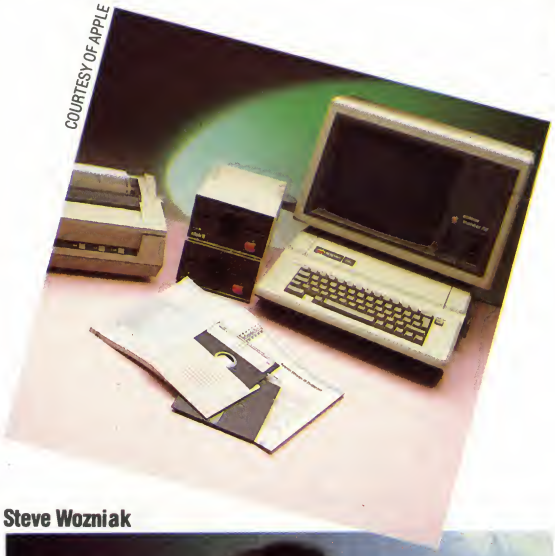
Steve Jobs

The Company's Core
Steve Wozniak designed and built the first Apple 1 (an uncased PCB) in his garage. When the design was modified and encased, creating the Apple II, his friend Steve Jobs turned Apple into the commercial success that it now is

In 1972 a young man called Gary Kildall was a consultant to Intel. His firm, Microprocessor Application Associates, was working on a computer language that Intel engineers could use to write software for the new microprocessor chips that Intel was manufacturing. Kildall thought it possible to link up a microprocessor with memory to an 8in floppy disk drive and to a teletype, in order to give each engineer a computer of his own. But Intel preferred to continue its practice of sharing a mainframe machine among its engineers.

Kildall and his friend John Torode, in another Californian garage, put together a system themselves. Torode built the hardware to make the floppy disk work with the processor, and Kildall wrote the software that enabled the processor to handle the disk. The program was called CP/M (Control Program/Microcomputers), a name derived from Kildall's work with Intel's programming language, which was called PL/M (Programming Language/Microcomputers).

The first disk operating system for micros was taken up quickly by hardware manufacturers wanting to put disk drives on their machines. The software influenced design too: CP/M would run only on the 8080 and faster 8085 processors from Intel, and on the similar Z80 from Zilog. The Z80 became the standard chip for any CP/M machine, and CP/M compatibility the goal for software firms.



COURTESY OF APPLE

Steve Wozniak

Gary Kildall
The latest operating systems are developed by large teams of programmers, but CP/M was written by Gary Kildall single-handedly. Even some of the later versions reflected the fact that it was developed for very crude hardware

COURTESY OF OSBORNE

Adam Osborne
Described by some as a 'poacher-turned-gamekeeper', Adam Osborne was for many years a leading microcomputer journalist, before starting his own company and producing the world's first portable computer

Apart from operating systems, home computers needed a programming language in which people could write their programs. BASIC, developed at Dartmouth College, USA, as an easy-to-learn language, was an obvious choice.

Bill Gates, a graduate in Seattle, produced a BASIC interpreter for micros, a translation program

that fitted in a limited-memory chip and could be incorporated into a home machine. Gates' company, Microsoft, became as much the standard producer in languages as Digital Research became in operating systems, and his fortune was made.

With these developments, advances in hardware and applications software quickly followed. Dan Bricklin and Bob Frankston produced the first micro spreadsheet program, VisiCalc, at their Software Arts company. Distributed by Personal Software on the Apple II, this became the best-selling applications package ever, and to emphasise its connection Personal Software changed its name to VisiCorp. WordStar was produced by Seymour Rubinstein's MicroPro and became the major best-seller in the CP/M word processor market.

The hardware that these packages were running on became cheaper and more powerful. Adam Osborne, who began as a technical writer, journalist and software publisher after moving to the US from Britain, launched a successful business computer with a large amount of expensive software included in the already

COURTESY OF HEWLETT PACKARD

Small Beginnings
Surprisingly, the technology of microcomputers developed more from the sophisticated programmable calculators (such as this Hewlett-Packard HP65) than from the earlier generation of minicomputers

Herman Hauser

JUDY GOLDHILL

From Little Acorns. . .
Though less innovative in price than Sinclair, the contribution of Chris Curry and Herman Hauser (as designers and directors of Acorn computers) has been no less valid. The Acorn Atom, BBC Microcomputer, and the Electron are all seen as milestones in their own right

JUDY GOLDHILL

Chris Curry

Sir Clive Sinclair
Following his innovative products in hi-fi, calculators, miniature radios, pocket TVs and digital watches, the unparalleled success of his microcomputers (ZX80, ZX81, and the Spectrum) earned him a knighthood in 1983

COURTESY OF SINCLAIR RESEARCH

COURTESY OF SINCLAIR RESEARCH

IAN McKINNELL

The Big One
IBM's acceptance of the microcomputer's viability didn't come until 1982, but it still had the predicted effect. Almost every new business microcomputer now boasts IBM PC compatibility to capitalise on the huge base of software

competitive price. And, of course, there is Sir Clive Sinclair, who set new price levels with the ZX80, ZX81 and ZX Spectrum, and has made home computing possible for millions of first-time users.

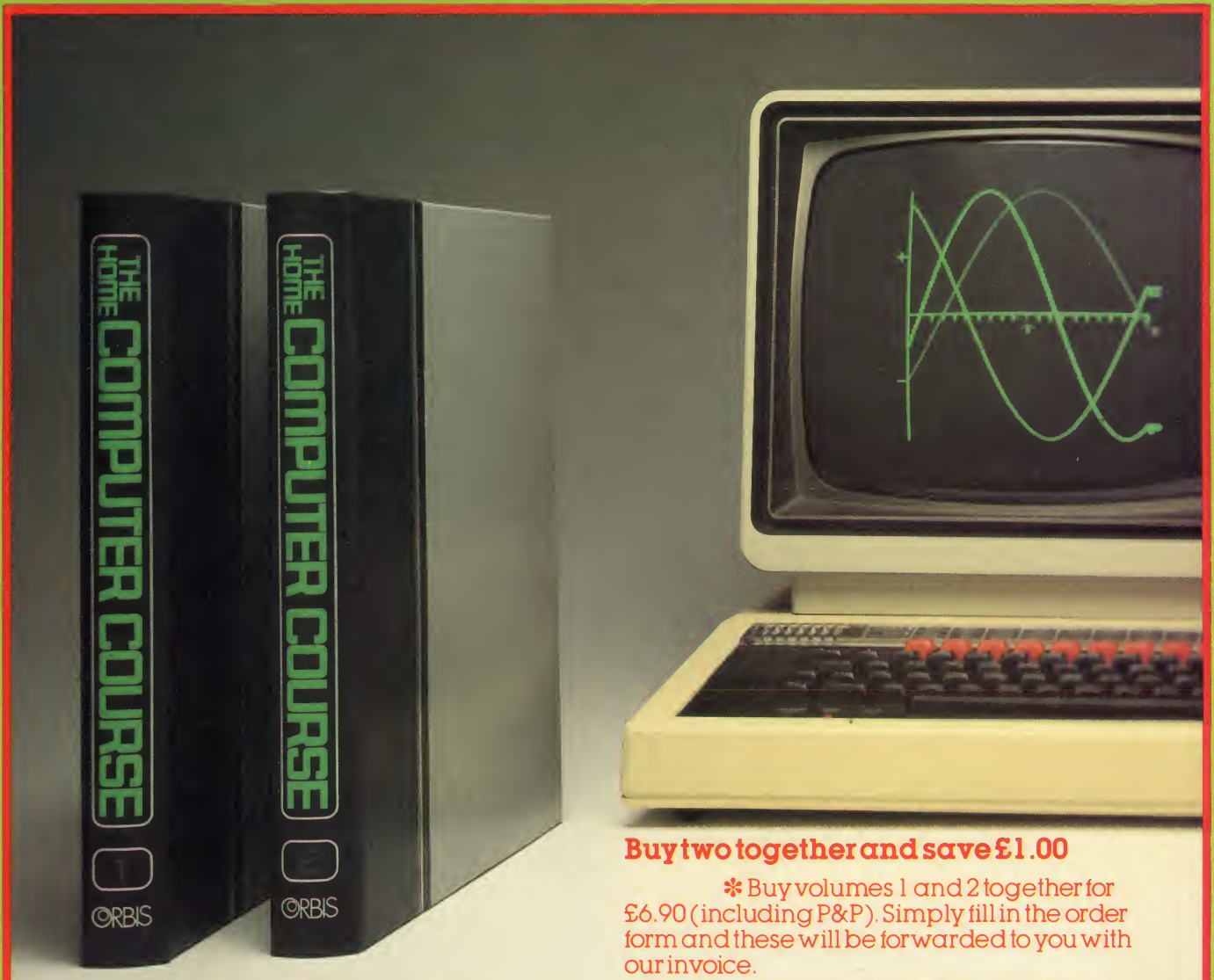
The standard for microcomputers in the last two years has been set by IBM with the IBM PC. Launched in 1982, this machine is proving increasingly popular. Virtually every software house and hardware peripherals maker is now producing material for the PC, and that in turn is encouraging more and more people to choose the machine.

The IBM PC brings together several of the pioneers from the early days of the micro industry. The microprocessor comes fom Intel, the originator of that technology; the operating systems come from Bill Gates' Microsoft, diversifying from languages, and from Gary Kildall's Digital Research; and two of the first software packages put on the machine were VisiCalc and WordStar.

Steve Wozniak and Steve Jobs still run Apple, on the whole in direct competition with IBM, and are pinning their company's hopes on the revolutionary technology in the Lisa (see page 261) and Macintosh (a cut-down version of the Lisa at around £2,000). Chuck Peddle started his own company, Sirius, and took a big slice of the UK business before IBM arrived, though his company has since encountered financial difficulties.

But Peddle will surely be back. The short history of micro business shows that the originators are also the survivors — even when the multinationals try to take over the game.

THE HOME COMPUTER COURSE BINDER



Now that your collection of Home Computer Course is growing, it makes sound sense to take advantage of this opportunity to order the two specially designed Home Computer Course binders.

The binders have been commissioned to store all the issues in this 24 part series.

At the end of the course the two volume binder set will prove invaluable in converting your copies of this unique series into a permanent work of reference.

Buy two together and save £1.00

✻ Buy volumes 1 and 2 together for £6.90 (including P&P). Simply fill in the order form and these will be forwarded to you with our invoice.

✻ If you prefer to buy the binders separately please send us your cheque/postal order for £3.95 (including P&P). We will send you volume 1 only. Then you may order volume 2 in the same way – when it suits you!

Overseas readers: This binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in Issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain their binders **now**. For details please see inside the front cover.

Binders may be subject to import duty and/or local tax.

THE LAST WORD IN LOGIC

FREE WITH THIS ISSUE

ISSUE ONE of

THE HOME COMPUTER ADVANCED COURSE

FREE NEXT WEEK

in The Home Computer Advanced Course Issue Two FREE Game Cassette

Order The Home Computer Advanced Course Issue 2 from your Newsagent Now